
Frequently Asked Questions About JBossCache

Ben Wang <ben.wang@jboss.com>

Bela Ban <bela@jboss.com>

June, 2005

This is frequently asked questions regarding to JBossCache.

1. General Information

Q: What is JBossCache?

A: JBossCache is a replicated and transactional cache. It is replicated since multiple JBossCache instances can be distributed (within the same JVM or across several JVMs whether they reside on the same machine or on different machines on a network) and data is replicated across the whole group. It is transactional because a user can configure a JTA transaction manager and make the cache operation transactional. Note that the cache can also be run without any replication; this is the local mode.

Currently, it consists of two components: `TreeCache` and `TreeCacheAop`. `TreeCache` is a tree-structured cache that provides replication and transaction context, while `TreeCacheAop` extends the functionality of `TreeCache` but behaves as a true object cache providing transparent and finer-grained object mapping into internal cache.

Q: Who are the JBossCache developers?

A: JBossCache has been developed by Bela Ban, Ben Wang, Harald Gliebe, Manik Surtani and Brian Stansberry. Currently, Bela is the overall lead while Ben leads JBossCacheAop.

Q: What is the license for JBossCache?

A: JBossCache is licensed under LGPL.

Q: Where can I download the package?

A: You can either obtain it from JBoss cvs repository in SourceForge.net, or the standalone package also in SourceForge.net.

Q: How do I build the JBossCache myself from cvs?

A: JBossCache now is a separate standalone project from jboss-head. Check <http://www.jboss.org/wiki/Wiki.jsp?page=CVSRepository> for details. The module name is JBossCache. To build, do `sh build.sh jar`. This will produce `jboss-cache.jar` in the `dist/lib` directory. Or if you want to build the standalone package, do `sh build.sh dist` this will produce `dist/jboss-cache-dist.zip`.

Q: What JVM is supported by JBossCache?

A: JBossCache has been tested and supported on JDK1.4 and JDK5.0. On jboss-3.2 cvs tree, it also compiles on JDK1.3, but there is no official support for it though.

Q: How do I know the version of JBossCache that I am using?

A: Since release 1.2, you can check the jar version by running: `java -jar jboss-cache.jar org.jboss.cache.Version`.

Q: Can I run JBossCache outside of JBoss Application Server?

A: Of course! JBossCache comes with two flavors:

- Integrated with JBoss server as a MBean service.
- As a standalone that can run outside of JBoss server such as under Weblogic or Websphere. Of course, it can also run under a standalone Java process (i.e., outside J2EE context).

Q: Where can I report bugs or problems?

A: Please report any bugs or problems to JBossCache forum [<http://www.jboss.org/index.html?module=bb&op=viewforum&f=157>].

2. JBossCache in general and TreeCache

Q: How do I deploy JBossCache as a MBean service?

A: To deploy JBossCache as a MBean inside JBoss, you can copy the configuration xml file over to the deploy directory (from all configuration whereby the necessary jars are present). Under the standalone package `etc/META-INF` directory, there are example configuration files for different cache modes that can be used to deploy JBossCache as well.

Q: How do I know if my JBossCache MBean has been deployed?

A: To verify that your JBossCache MBean is deployed correctly, you can first check the log output under the

command console. Next you can verify it from JBoss JMX console. Look for `jboss.cache` domain.

Q:
How do I access the JBossCache MBean?

A:
Accessing the JBossCache MBean is just like accessing any JBoss MBean. Here is a code snippet:

```
import org.jboss.mx.util.MBeanServerLocator;
import javax.management.MBeanServer;
import javax.management.ObjectName;
...

public init() throws Exception {
    try {
        cacheService = new ObjectName("jboss.cache:service=TreeCache");
        server = MBeanServerLocator.locate();
    } catch (Exception ex) {
        throw new CreateException(ex.toString());
    }
}

public Object get(String fqcn, String key) throws Exception {
    return server.invoke(cacheService, "get",
        new Object[]{fqcn, key},
        new String[]{String.class.getName(),
            Object.class.getName()});
}
```

Q:
Can I run JBossCache on 3.2 releases?

A:
Yes. The JBossCache source code is also up to date on the `jboss-3.2` cvs branch. However, only `TreeCache` is supported there since `JBossAop` lives only under `jboss-head`.

Q:
Can I run multiple JBossCache instances on the same VM?

A:
Yes. There are couple scenarios that you want to run multiple instances of JBossCache. For example, you want to run multiple local cache instances with each instance has their own configuration (e.g., different cache policy). In this case, you will need to multiple xml configuration files.

Q:
Can TreeCache run as a second level cache inside Hibernate?

A:
Yes. Since Hibernate 2.1 release, you can configure it to use JBossCache (namely, `TreeCache`) as a second level cache. For details, see [Hibernate documentation](#).

Note that since `Hibernate3.0.2` and `JBossCache1.2.2`, we have fixed a critical bug that depends on the usage pattern can cause deadlock during query caching.

Q:
How can I configure JBossCache?

- A:
You can configure the JBossCache through a configuration xml file. Or you can set it programatically through its get/set methods. Check with the documentation for both examples.
- Q:
In the configuration xml file, there are tags such as `class`, `MBean`, etc. What are these?
- A:
These are tags for deploying JBossCache as a JBoss MBean service. For reason of compatibility, we have kept them in the standalone package as well, specifically, the `MBean` tag. If you run in standalone mode, JBossCache will ignore these elements.
- Q:
What is the difference between the different cache modes?
- A:
JBossCache has three different cache modes, i.e., `LOCAL`, `REPL_SYNC`, and `REPL_ASYNC`. If you want to run JBossCache as a single instance, then you should set the cache mode to `LOCAL` so that it won't invoke replication layer. If you want to have synchronous replication among different JBossCache instances, you set it to `REPL_SYNC`. Finally, for asynchronous replication, use `AYSNC_REPL`.
- Note that `ASYNC_REPL` is non-blocking. This can be useful when you want to have another JBossCache serving as a mirror or backup.
- Q:
How does JBossCache's replication mechanism work?
- A:
JBossCache leverage JGroups as a replication layer. A user can configure the cluster of JBossCache instances by sharing the same cluster name (`cluster name`). There is also an option of whether to populate the cache data upon starting a new instance in the `ClusterConfig` attribute.
- Note that once all instances join the same replication group, every replication change is propagated to all participating members. There is no mechanism for sub-partitioning where some replication can be done within only a subset of members. This is in our to do list.
- Q:
If I have the need for different treecache properties (e.g., `CacheMode` and `IsolationLevel`), do I simply need to create multiple treecache instances with the appropriate configuration?
- A:
Yes. All the above mentioned properties are per cache instance. Therefore you will need a separate JBossCache instance.
- Q:
Does the TreeCache config `clusterName` have any relation to the JBoss cluster `PartitionName`?
- A:
Yes. They are both JGroups group name. Besides the notion of channel is JGroups, it also can partition the channel into different group names.
- Q:
When using multiple JGroups based components [`cluster-service.xml`, `treecache` (multiple instances)], what is the correct/valid way to configure those components to make sure my multicast addresses don't conflict?

- A:
There are two parameters to consider: multicast address (plus port) and the group name. At minimum, you will have to run components on different group name. But whether to run them on the same channel depends upon whether the communication performance is critical for you or not. If it is, then it'd be the best to run them on different channels.
- Q:
Does JBossCache currently supports cache persistence storage?
- A:
Yes. Starting with release 1.1, JBossCache has a CacheLoader interface that supports the cache persistency. See below.
- Q:
Does JBossCache currently supports cache passivation/ overflow to a data store?
- A:
Yes. Starting with release 1.2.4, JBossCache uses the CacheLoader to support cache passivation/ overflow. See documentation on how to configure and use this feature.
- Q:
Is JBossCache thread safe?
- A:
Yes. It is.
- Q:
Does JBossCache support XA (2PC) transaction now?
- A:
No, although it is also on our to do list. Our internal implementation does use a similar 2PC procedure to coordinate transaction among different instances.
- Q:
What kind of TransactionManager is supported by JBossCache?
- A:
JBossCache supports any TransactionManager that is JTA compliant such as JBossTM. A user can configure the transaction manager through the configuration xml setting. JBossCache also has a built in dummy transaction manager (`org.jboss.cache.tm.DummyTransactionManager`) for testing purpose only. But note that `DummyTransactionManager` is not thread safe though. I.e., it does not support concurrent transaction. Instead, only one transaction is allowed at one time.
- Q:
How do I setup the cache to be transactional?
- A:
You either use the default (JBoss) TransactionManager to run JBossCache inside JBoss, or you have to implement the `TransactionManagerLookup` interface, and return an instance of your `javax.transaction.TransactionManager`. The configuration property `TransactionManagerLookupClass` defines the class to be used by the cache to fetch a reference to a `TransactionManager`. It is trivial to implement this class to support other TransactionManagers. Once this attribute is specified, the cache will look up the transaction context from this transaction manager.

For the client code, here is a snippet to start and commit a transaction:

```
tx = (UserTransaction)new InitialContext(prop).lookup("UserTransaction");
tree = new TreeCache();
config = new PropertyConfigurator();
config.configure(tree, "META-INF/replSync-service.xml");

tx.begin()
tree.put(fqn, key, value);
tx.commit();
```

Q: How do I control the cache locking level?

A: JBossCache let you control the cache locking level through the transaction isolation level. This is configured through the attribute `IsolationLevel`. Currently, JBossCache employs pessimistic locking internally. And the transaction isolation level from the pessimist locking corresponds to JDBC isolation levels, namely, `NONE`, `READ_UNCOMMITTED`, `READ_COMMITTED`, `REPEATABLE_READ`, and `SERIALIZABLE`. For details, please refer to the user manual.

Q: How does the write lock applies to a fqn node, say, `"/org/jboss/test"`?

A: First of all, JBossCache has a notion of root that serves as a starting point for every navigational operation. The default is `"/"` (since the default separator is `"/"` for the fqn). The locking then is applied to the node under root, for example `"/org"` (no locking `"/"`).

Furthermore, let's say when JBossCache needs to apply a write lock on node `"/org/jboss/test"`, it will first try to obtain read lock from the parent nodes recursively (in this example, `"/org"`, and `"/org/jboss"`). Only when it succeeds then it will try to obtain a write lock on `"/org/jboss/test"`.

Q: Can I use the cache locking level even without a transaction context?

A: Yes. JBossCache controls the individual node locking behavior through the isolation level semantics. This means even if you don't use a transaction, you can specify the lock level via isolation level. You can think of the node locking behavior outside of a transaction as if it is under transaction with `auto_commit` on.

Q: During replication mode (e.g., `REPL_SYNC` and `REPL_ASYNC`), how often does the cache do the replication?

A: If the updates are under transaction, then the replication happens only when the transaction is about to commit (actually during the prepare stage internally). That is, it will be a batch update. However, if the operations are not under transaction context, then each update will trigger replication. Note that this has performance implication if network transport is heavy (it usually is).

Q: How can do a mass removal?

A: If you do a `cache.remove("/root")`, it will recursively remove all the entries under `"/root"`.

Q: Can I monitor and manage the JBossCache?

A: If you are running JBossCache as a MBean service inside JBoss, you can monitor and manage through the jmx console. We plan to add more attributes such as cache hit/miss numbers in the future release.

Q: Does JBossCache support partitioning?

A: Not right now. JBossCache does not support partitioning that a user can configure to have different set of data residing on different cache instances while still participating as a replication group.

Q: Does JBossCache handle the concept of application classloading inside, say, a J2EE container?

A: Application-specific classloading is used widely inside a J2EE container. For example, a web application may require a new classloader to scope a specific version of the user library. However, by default JBossCache is agnostic to the classloader. In general, this leads to two kinds of problems:

- Object instance is stored in cache1 and replicated to cache2. As a result, the instance in cache2 is created by the system classloader. The replication may fail if the system classloader on cache2 does not have access to the required class. Even if replication doesn't fail, a user thread in cache2 may not be able to access the object if the user thread is expecting a type defined by the application classloader.
- Object instance is created by thread 1 and will be accessed by thread 2 (with two different classloaders). JBossCache has no notion of the different classloaders involved. As a result, you will have a `ClassCastException`. This is a standard problem in passing an object from one application space to another; JBossCache just adds a level of indirection in passing the object.

To solve the first kind of issue, in 1.2.4 JBossCache introduced the concept of a `TreeCacheMarshaller`. Basically, this allows application code to register a classloader with a portion of the cache tree for use in handling objects replicated to that portion. See the `TreeCacheMarshaller` section of the user guide for more details.

To solve the second kind of issue, the only solution (that we know of) is to cache "serialized" byte code and only de-serialize it during every object get (and this will be expensive!). That is, during a put operation, the object instance will be serialized and therefore can be deserialized safely by a "foreign" classloader. However, the performance penalty of this approach is quite severe so in general another local in-vm version will need to be used as a "near-line" cache. Note also that each time the serialized bytes are deserialized, a new instance of the object is created.

To help with this kind of handling, JBoss has a utility class called `MarshaledValue` that wraps around the serialized object. Here is a code snippet that illustrates how you can create a wrapper around JBossCache to handle the classloader issue:

```
import org.jboss.invocation.MarshaledValue;
```

```
public class CacheService {
    private TreeCache cache_;

    public Object get(Fqn fqn, String key) {
        return getUnMarshaledValue(cache_.get(fqn, key));
    }

    public Object set(Fqn fqn, String key, Object value) {
        cache_.put(fqn, key, getMarshaledValue(value));
        return value; // only if successful
    }

    ...

    private Object getUnMarshaledValue(Object value) {
        // assuming we use the calling thread context classloader
        return ((MarshaledValue)value).get();
    }

    private Object getMarshaledValue(Object value) {
        return new MarshaledValue(value);
    }
}
```

Q: Does JBossCache currently support pre-event and post-event notification?

A: Yes. Starting with release 1.2.4, JBossCache has introduced `ExtendedTreeCacheListener` which takes in consideration pre and post event notification. See documentation for more details. Note that `TreeCacheListener` and `ExtendedTreeCacheListener` will be merged into `TreeCacheListener` in release 1.3.

Q: How do I implement a custom listener to listen to `TreeCache` events?

A: You create a class (`myListener`) that extends `AbstractTreeCacheListener` and provide concrete implementation for the node events that you are interested in. Then you add this listener to the `TreeCache` instance on startup to listen to the events as they occur by calling `TreeCache.addTreeCacheListener(myListener)`.

```
public class MyListener extends AbstractTreeCacheListener
{
    ...

    public void nodeModify(Fqn fqn, boolean pre, boolean isLocal) {
        if(log.isTraceEnabled()){
            if(pre)
                log.trace("Event DataNode about to be modified: " + fqn);
            else
                log.trace("Event DataNode modified: " + fqn);
        }
    }

    ...
}
```

3. TreeCacheAop

Q: What is TreeCacheAop?

A: TreeCacheAop (currently implemented as a sub-class of TreeCache) is a fine-grained field-level replicated and transactional "object-oriented" cache. By "object-oriented", we mean that the cache: 1) automatically manages object mapping and relationship for a client under both local and replicated cache mode, 2) provides support for inheritance relationship between "aspectized" POJOs. By leveraging the dynamic AOP in JBossAop, it is able to map a complex object into the cache store, preserve and manage the object relationship behind the scene. During replication mode, it performs fine-granularity (i.e., on a per-field basis) update, and thus has the potential to boost cache performance and minimize network traffic.

From a user perspective, once the object is managed by the cache, all cache operations are transparent. Therefore, all the usual in-VM POJO method semantics are still preserved, providing ease of use. For example, if a POJO has been put in TreeCacheAop (by calling putObject, for example), then any get/set method will be intercepted by TreeCacheAop to provide the data from the cache.

Q: Does TreeCacheAop have all the functional capabilities of TreeCache?

A: Yes. TreeCacheAop extends TreeCache so it has all the same features TreeCache such as cache mode, transaction isolation level, and eviction policy.

Q: Can I run TreeCacheAop in JBoss3.2.x application server?

A: Yes and no. Yes, since JBossAop can also be back-ported to 3.2.x (see JBossAop wiki for details). However, it will take some effort. Therefore, the recommended JBoss version is 4.x to run TreeCacheAop.

Q: Can TreeCacheAop run as a MBean as well?

A: Yes. It is almost the same as TreeCache MBean. The only difference is the object name and the class name. E.g., instead of

```
<mbean code="org.jboss.cache.TreeCache" name="jboss.cache:service=TreeCache">
```

```
<mbean code="org.jboss.cache.aop.TreeCacheAop" name="jboss.cache:service=TreeCacheAop">
```

Q: What is the difference between TreeCache and TreeCacheAop?

A: Think of TreeCacheAop as a TreeCache on steroid. :-) Seriously, both are cache store. However, while TreeCache only provides pure object reference storage (e.g., put(FQN fqN, Objecy key, Object value)), TreeCacheAop goes beyond that and performs fine-grained field level replication object mapping and relationship management for a user behind the scene. As a result, if you have a complex object systems that you

would like to cache, you can have TreeCacheAop manages for you. You simply treat your object systems as they are residing in-memory, e.g., use your regular POJO methods without worrying about cache management. Furthermore, this is true in replication mode as well.

Q:

Can I pre-compile the aop classes such that I don't need to use the system classloader and jboss-aop configuration xml?

A:

Yes. The latest versions of JBossCache has a pre-compiler option called aopc. You can use this option to pre-compile your "aspectized" POJO. Once the classes have been byte code generated, it can be treated as regular class files, i.e., you will not need to include any `jboss-aop.xml` that specifies the advisable POJO and to specify the JBossAop system class loader.

For an example of how to use aopc, please see the `build.xml` in the standalone package (look for the aopc Ant target).

Q:

How do use aopc on multiple module directories?

A:

In aopc, you specify the src path for a specific directory. To pre-compile multiple ones, you will need to invoke aopc multiple times.

Q:

What's in the `jboss-aop.xml` configuration?

A:

`jboss-aop.xml` is needed for POJO instrumentation. In `jboss-aop.xml`, you can declare your POJO (e.g., Person) to be "prepared", a JBOssAop term to denote that the object will be "aspectized" by the system. After this declaration, JBossAop will invoke any interceptor that associates with this POJO. TreeCacheAop will dynamically add an `org.jboss.cache.aop.CacheInterceptor` to this POJO to perform object mapping and relationship management.

Note that to add your POJO, you should declare all the fields to be "prepared" as in the example.

Q:

Can I use annotation instead of the xml declaration?

A:

Yes, starting with JBossCache1.2.3, you can use annotation to instrument your POJO. Check the documentation for details. However, keep in mind, JBossCache only supports JDK1.4 so far. So JDK5.0 style annotation is not fully supported yet. As a result, you will need a JDK1.4 annotation pre-compiler provided by JBossAop.

Q:

Is it possible to store the same object multiple time but with different FQN paths? Like `/foo/byName` and `/foo/byId` ?

A:

Yes, you can use TreeCacheAop to do that. It supports the notion of object reference. TreeCacheAop manages the unique object through association of the dynamic cache interceptor.

Q: Do I need to declare all my objects "prepared" in `jboss-aop.xml`?

A: Not necessarily. If there is some objects that you don't need the cache to manage it for you, you can leave it out of the declaration. The cache will treat this object as a "primitive" type. However, the object will need to implement `Serializable` interface for replication though.

Q: Can the cache aop intercept update via reflection?

A: No. The update via reflection will not be intercepted in `JBossAop` and therefore `TreeCacheAop` will not be able to perform the necessary synchronization.

Q: When I declared my POJO to be "aspectized", what happens to the fields with transient, static, and final modifiers?

A: `TreeCacheAop` currently will ignore the fields with these modifiers. That is, it won't put these fields into the cache (and thus no replication either).

Q: What are those keys such as `JBoss:internal:class` and `AOPInstance`?

A: They are for internal use only. User should ignore these key and values in the node hashmap.

Q: What about Collection classes? Do I need to declare them "prepared"?

A: No. Since the Collection classes such as `ArrayList` are java util classes, aop by default won't instrument these classes. Instead, `TreeCacheAop` will generate a dynamic class proxy for the Collection classes (upon the `putObject` call is invoked). The proxy will delegate the operations to a cache interceptor that implements the actual Collection classes APIs. That is, the system classes won't be invoked when used in `TreeCacheAop`.

Internally, the cache interceptor implements the APIs by direct interaction with respect to the underlying cache store. Note that this can have implication in performance for certain APIs. For example, both `ArrayList` and `LinkedList` will have the same implementation. Plan is currently underway to optimize these APIs.

Q: How do I use `List`, `Set`, and `Map` dynamic proxy?

A: `TreeCacheAop` supports classes extending from `List`, `Set`, and `Map` without users to declare them "aspectized". It is done via a dynamic proxy. Here is a code snippet to use an `ArrayList` proxy class.

```
ArrayList list = new ArrayList();
list.add("first");

cache.putObject("/list/test", list); // Put the list under the aop cache
list.add("second"); // Won't work since AOP intercepts the dynamic proxy not the original POJO.

ArrayList myList = (List)cache.getObject("/list/test"); // we are getting a dynamic proxy instead
```

```
myList.add("second"); // it works now
myList.add("third");
myList.remove("third");
```

Q: What is the proper way of assigning two different keys with Collection class object?

A: Let's say you want to assign a List object under two different names, you will need to use the class proxy to insert the second time to ensure both are managed by the cache. Here is the code snippet.

```
ArrayList list = new ArrayList();
list.add("first");

cache.putObject("/list", list); // Put the list under the aop cache

ArrayList myList = (List)cache.getObject("/list"); // we are getting a dynamic proxy instead
myList.add("second"); // it works now

cache.putObject("/list_alias", myList); // Note you will need to use the proxy here!!
myList.remove("second");
```

Q: OK, so I know I am supposed to use proxy when manipulating the Collection classes once they are managed by the cache. But what happens to Pojos that share the Collection objects, e.g., a List instance that is shared by 2 Pojos?

A: Pojos that share Collection instance references will be handled by the cache automatically. That is, when you ask the Cache to manage it, the Cache will dynamically swap out the regular Collection references with the dynamic proxy ones. As a result, it is transparent to the users.

Q: What happens when my "aspectized" POJO has field members that are of Collection class ?

A: When a user puts a POJO into the cache through the call putObject, it will recursively map the field members into the cache store as well. When the field member is of a Collection class (e.g., List, Set, or Map), TreeCacheAop will first map the collection into cache. Then, it will swap out dynamically the field reference with an corresponding proxy reference.

This is necessary so that an internal update on the field member will be intercepted by the cache.

4. Eviction Policy

Q: Does JBossCache support eviction policy?

A: Yes. JBossCache currently implements a LRU eviction policy for both TreeCache (org.jboss.cache.eviction.LRUPolicy) and TreeCacheAop (org.jboss.cache.eviction.AopLRUPolicy). Users can also plug in their own eviction policy algorithms. See user manual for details. Currently there is user-contributed policy called FIFOPolicy that evicts the node

based on FIFO principle only.

Q: Why can't I use `org.jboss.cache.eviction.LRUPolicy` for `TreeCacheAop` as well?

A: For `TreeCacheAop`, you will need to use `org.jboss.cache.eviction.AopLRUPolicy` because AOP has its eviction algorithm, although is LRU but has totally different notion of an "object", for example.

Q: Does JBossCache's implemented LRU eviction policy operates in replication mode?

A: Yes and no. :-)

The LRU policy only operates in local mode. That is, nodes that are only evicted locally. This may cause the cache contents not to be synchronized temporarily. But when a user tries to obtain a cache content of an evicted node and finds out that is null (e.g., `get` returns null), it should get it from other data source and re-populate the data in the cache. During this moment, the node content will be propagated and the cache content will be in sync.

However, you still can run eviction policy with cache mode set to either `REPL_SYNC` or `REPL_ASYNC`. Depends on your use case, you can set multiple cache instances to have their own eviction policy (of which is operated locally) or just have selected instance with eviction policy activated.

Also note that, with cache loader option, an locally evicted node can also be persisted to the backend store and a user can retrieve it from the store later on.

Q: Does JBossCache support Region?

A: Yes. JBossCache has the notion of region where a user can configure the eviction policy parameters (e.g., `maxNodes` or `timeToIdleSeconds`)

A region in JBossCache denotes a portion of tree hierarchy, e.g., a fully qualified name (FQN). For example, a user can define `/org/jboss` and `/org/foocom` as two separate regions. But note that you can configure the region programmatically now, i.e., everything has to be configured through the xml file.

Q: What are the `EvictionPolicyConfig` tag parameters for `org.jboss.cache.eviction.LRUPolicy`?

A: They are:

Table 1. Parameters

<code>wakeUpIntervalInSeconds</code>	Interval where the clean up thread wakes to process the sitting queue and sweep away the old data.
<code>region</code>	A area where each eviction policy parameters are specified. Note that it needs a minimum of

	/_default region.
maxNodes	Max number of nodes allowed in the eviction queue. 0 means no limit.
timeToLiveInSeconds	Age (in seconds) for the node to be evicted in the queue. 0 denotes no limit.

Q: I have turned on the eviction policy, why do I still get "out of memory" (OOM) exception?

A: OOM can happen when the speed of cache access exceeds the speed of eviction policy handling timer. Eviction policy handler will wake up every `wakeupIntervalInSeconds` seconds to process the eviction event queue. And the queue size is fixed at 20000 now. So when the queue size is full, it will create a backlog and cause OOM to happen unless the eviction timer catches up. To address this problem, in addition to increase the VM heap size, you can also reduce the `wakeupIntervalInSeconds` so the timer thread processes the queue more frequently.

We will also externalize the queue size so it will be configurable in the next release.

5. CacheLoader

Q: What is a CacheLoader?

A: A CacheLoader is the connection of JBossCache to a (persistent) data store. The CacheLoader is called by JBossCache to fetch data from a store when that data is not in the cache, and when modifications are made to data in the cache the CacheLoader is called to store those modifications back to the store.

In conjunction with eviction policies, JBossCache with a CacheLoader allows a user to maintain a bounded cache for a large backend datastore. Frequently used data is fetched from the datastore into the cache, and the least used data is evicted, in order to provide fast access to frequently accessed data. This is all configured through XML, and the programmer doesn't have to take care of loading and eviction.

JBossCache currently ships with 3 CacheLoader implementations:

- **FileCacheLoader:** this implementation uses the file system to store and retrieve data. JBossCache nodes are mapped to directories, subnodes to subdirectories etc. Attributes of a node are mapped to a file data inside the directory.
- **BdbjeCacheLoader:** this implementation is based on the Sleepycat Java Edition database, a fast and efficient transactional database. It uses a single file for the entire store. Note that if you use Sleepycat's CacheLoader with JBossCache, and ship your product, you have to acquire a commercial license from Sleepycat, see <http://www.sleepycat.com/jeforjboss-cache> for details.
- **JDBCCacheLoader:** this implementation uses the relational database as the persistent storage.

Q: Can writing to CacheLoaders be asynchronous?

A: As of JBossCache 1.2.4, yes. Set the `CacheLoaderAsynchronous` property to true. See the JBossCache documentation for a more detailed discussion. By default though, all cache loader writes are synchronous and will block.

Q: Can I write my own CacheLoader ?

A: Yes. A CacheLoader is a class implementing `org.jboss.cache.loader.CacheLoader`. It is configured via the XML file (see JBossCache and Tutorial documentation).

Q: Does a CacheLoader have to use a persistent store ?

A: No, a CacheLoader could for example fetch (and possibly store) its data from a webdav-capable webserver. Another example is a caching proxy server, which fetches contents from the web. Note that an implementation of CacheLoader may not implement the 'store' functionality in this case, but just the 'load' functionality.

Q: What can I use a CacheLoader for?

A: Some applications:

- HTTP sessions can be persisted (besides being replicated by JBossCache). The CacheLoader can be configured to be shared, or unshared, meaning that every node in a cluster has its own local store. It is also possible to attach a CacheLoader to just *one* of the nodes.
- Simple persistence for POJOs. Use of JBossCache aop and a local CacheLoader persist POJOs transparently into the store provided by the CacheLoader.
- Highly available replicated and persisted data store. The service is up as long as at least 1 node is running, but even if all nodes are taken offline, when the first node is started again, the data previously saved will still be available (e.g. a shopping cart).
- A caching web proxy (a la Squid): all data are contents of URLs, users access the proxy, and if the URL is not in the cache, the CacheLoader fetches it from the web. This could actually be a replicated and transactional version of Squid.

Q: How do I configure JBossCache with a CacheLoader?

A: Through XML: both the fully-qualified classname of the CacheLoader and its configuration string have to be given. JBossCache will then instantiate a CacheLoader. See JBossCache documentation for details.

Q:

Do I have to pay to use Sleepycat's CacheLoader?

A:

Not if you use it only for personal use. As soon as you distribute your product with BdbjeCacheLoader, you have to purchase a commercial license from Sleepycat. See details at <http://www.sleepycat.com/jeforjboss-cache>.